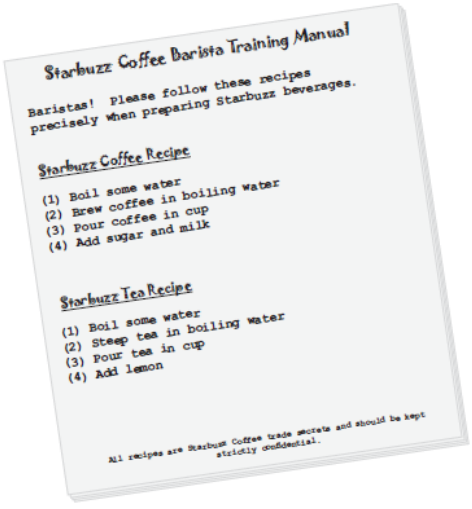
HoGent

BEDRIJF
EN
ORGANISATIE

Template Method Pattern

Algoritmen afschermen

Tijd voor wat meer cafeïne



Starbuzz Coffee Barista Training Manual

Baristas! Please follow these recipes precisely when preparing Starbuzz beverages.

Starbuzz Coffee Recipe

(1) Boil some water
(2) Brew coffee in boiling water
(3) Pour coffee in cup
(4) Add sugar and milk

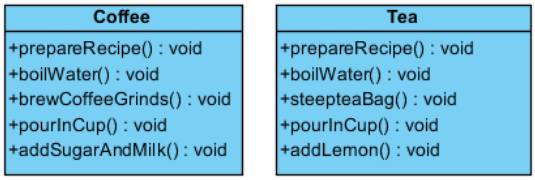
Starbuzz Tea Recipe

(1) Boil some water
(2) Steep tea in boiling water
(3) Pour tea in cup
(4) Add lemon

All recipes are Starbuzz Coffee trade secrets and should be kept strictly confidential.

De recepten voor koffie en thee lijken sterk op elkaar, niet?

Enkele koffie- en theeklassen



Enkele koffie- en theeklassen

```
public class Coffee {
  public void prepareRecipe() {
    boilWater();
    brewCoffeeGrinds();
    pourInCup();
    addSugarAndMilk();
  }
  public void boilWater() {
    System.out.println("Boiling water");
  }
  public void brewCoffeeGrinds() {
    System.out.println("Dripping coffee through filter");
  }
  public void pourInCup() {
    System.out.println("Pouring into cup");
  }
  public void addSugarAndMilk() {
    System.out.println("Adding sugar and milk");
  }}
```

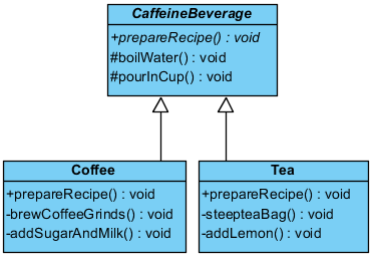
Code duplicatie ?

Code duplicatie?

```
public class Tea {
  public void prepareRecipe() {
    boilWater();
    steepteaBag();
    pourInCup();
    addLemon();
  }
  public void boilWater() {
    System.out.println("Boiling water");
  }
  public void steepteaBag() {
    System.out.println("Steeping the tea");
  }
  public void pourInCup() {
    System.out.println("Pouring into cup");
  }
  public void addLemon() {
    System.out.println("Adding lemon");
  }}
```

Koffie en thee abstraheren

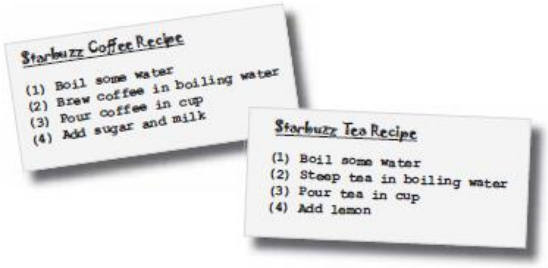
- Duplicate code -> ontwerp opschonen. Abstractie?



- Goed ontwerp?

Verder ontwikkelen van het ontwerp

- Wat hebben ze nog meer gemeen?



1. Wat water koken
2. Heet water gebruiken om koffie of thee te zetten
3. De betreffende drank in een kopje schenken
4. De juiste smaakstoffen aan de drank toevoegen

Verder ontwikkelen van het ontwerp

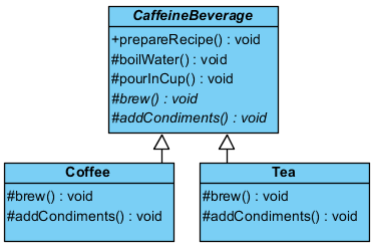
- prepareRecipe abstraheren?

```
public class Coffee extends CaffeineBeverage {
  public void prepareRecipe() {
    boilWater();
    brewCoffeeGrinds();
    pourInCup();
    addSugarAndMilk();
  }
```

brew()

addCondiments()

```
public class Tea extends CaffeineBeverage {
  public void prepareRecipe() {
    boilWater();
    steepteaBag();
    pourInCup();
    addLemon();
  }
```



CaffeineBeverage

```
public abstract class CaffeineBeverage {

    public final void prepareRecipe() {
        boilWater();
        brew();
        pourInCup();
        addCondiments();
    }

    protected final void boilWater() {
        System.out.println("Boiling water");
    }

    protected final void pourInCup() {
        System.out.println("Pouring into cup");
    }

    protected abstract void brew();

    protected abstract void addCondiments();
}
```

Nu wordt dezelfde methode gebruikt om zowel thee als koffie te zetten. Is als final gedeclareerd, omdat we niet willen dat onze subklasse, een override van deze methode kunnen uitvoeren en zo het recept kunnen wijzigen

Omdat koffie en thee deze methoden op verschillende manieren afhandelen, declareren we ze abstract!

De subklassen

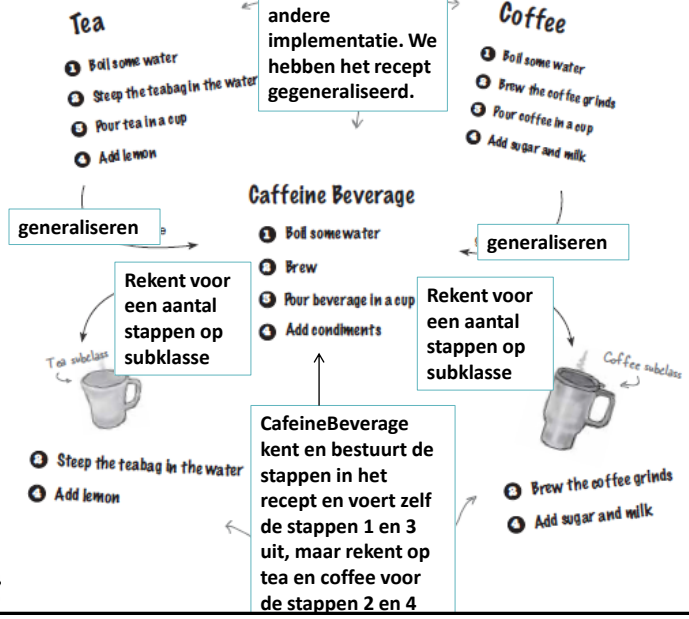
```
public class Coffee extends CaffeineBeverage {

  @Override
  protected void brew() {
    System.out.println("Dripping coffee through filter");
  }

  @Override
  protected void addCondiments() {
    System.out.println("Adding sugar and milk");
  }
}
```

```
public class Tea extends CaffeineBeverage {

  @Override
  protected void brew() {
    System.out.println("Steeping the tea");
  }

  @Override
  protected void addCondiments() {
    System.out.println("Adding lemon");
  }
}
```

HoGent

9

Wat hebben we gedaan?



HoGer

10

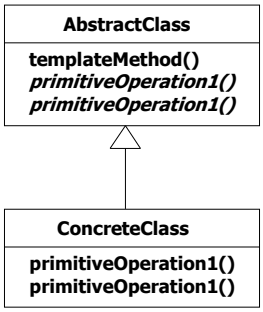
Koffie en thee maken

```
public static void main(String[] args) {
   System.out.println("Making  coffee");
   CaffeineBeverage beverage = new Coffee();
   beverage.prepareRecipe();
   System.out.println("Making  tea");
   beverage = new Tea();
   beverage.prepareRecipe();
}
```

```
Making  coffee
Boiling water
Dripping coffee through filter
Pouring into cup
Adding sugar and milk
Making  tea
Boiling water
Steeping the tea
Pouring into cup
Adding lemon
```

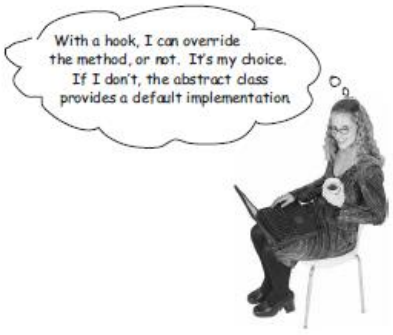

Template Method Pattern

Het Template Method Pattern definieert het skelet van een algoritme in een methode, waarbij sommige stappen aan subklassen worden overgelaten. De Template method laat subklassen bepaalde stappen in een algoritme herdefiniëren zonder de structuur van het algoritme te veranderen.

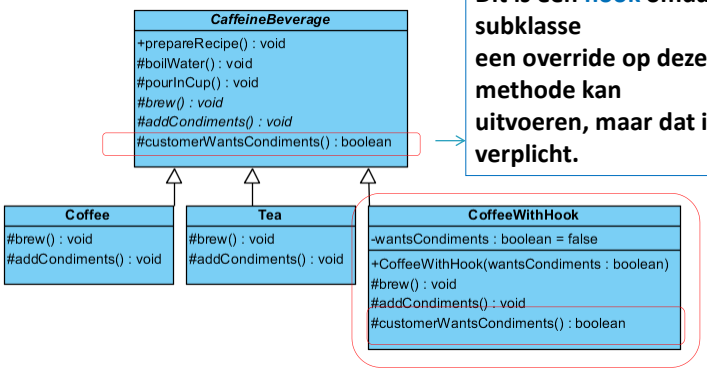


De templatemethode maakt gebruik van primitieve methoden om een algoritme te implementeren. Deze is echter ontkoppeld van de feitelijke implementatie van deze methoden.

Inhaken in een Template Method



De hook gebruiken



Dit is een hook omdat de subklasse een override op deze methode kan uitvoeren, maar dat is niet verplicht.

De hook gebruiken

- CaffeineBeverage klasse

```
public abstract class CaffeineBeverage {

    public final void prepareRecipe() {
        boilWater();
        brew();
        pourInCup();
        if (customerWantsCondiments()) {
            addCondiments();
        }
    }

    protected final void boilWater() {
        System.out.println("Boiling water");
    }

    protected final void pourInCup() {
        System.out.println("Pouring into cup");
    }

    protected abstract void brew();

    protected abstract void addCondiments();

    protected boolean customerWantsCondiments() {
        return true;
    }
}
```

We hebben een conditionele opdracht toegevoegd. Alleen voor de klant die smaakstoffen wil, roepen we de methode addCondiments op

De methode met de (bijna) lege standaard implementatie. Dit is de hook. Een subklasse kan een override uitvoeren, maar dit is niet verplicht

De hook gebruiken

- CoffeeWithHook klasse

```
public class CoffeeWithHook extends CaffeineBeverage {
    private boolean wantsCondiments;
    public CoffeeWithHook(boolean wantsCondiments){
        this.wantsCondiments = wantsCondiments;
    }
    @Override
    protected void brew() {
        System.out.println("Dripping coffee through filter");
    }
    @Override
    protected void addCondiments() {
        System.out.println("Adding sugar and milk");
    }
    protected boolean customerWantsCondiments()  {
        return wantsCondiments;
    }
}
```

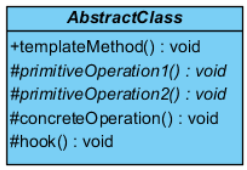
De koffie met de hook maken

```
public class Template {
  public static void main(String[] args) {
    System.out.println("Making  coffee");
    CaffeineBeverage beverage = new Coffee();
    beverage.prepareRecipe();
    System.out.println("Making  tea");
    beverage = new Tea();
    beverage.prepareRecipe();
    System.out.println("Making coffee with a hook");
    boolean answer = getUserInputForCoffee();
    beverage = new CoffeeWithHook(answer);
    beverage.prepareRecipe();
  }
  public static boolean getUserInputForCoffee() {
    String answer = null;
    System.out.println("Would you like milk and sugar with your coffee (y/n)?");
    Scanner in = new Scanner(System.in);
    return in.next().equalsIgnoreCase("y");
  }
```


De methodes in de abstracte klasse onder de loep

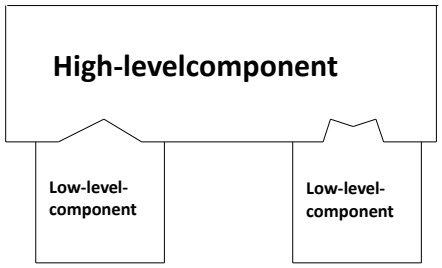
- We bekijken de soorten methoden die in een abstracte klasse kunnen bestaan wat verder

```
public abstract class AbstractClass {
  public final void templateMethod(){
    primitiveOperation1();
    primitiveOperation2();
    concreteOperation();
    hook();
  }
  //Deze zijn abstract en worden geimplementeerd door concrete subklassen
  protected abstract void primitiveOperation1();
  protected abstract void primitiveOperation2();
  //Een concrete implementatie, gedefineerd als final zodat geen override mogelijk is
  protected final void concreteOperation(){
    //implementatie
  }
  //Een concrete methode maar ze doet niets. Een hook
  protected void hook(){}
}
```



Het Hollywoodprincipe

Bel ons niet, wij bellen jou



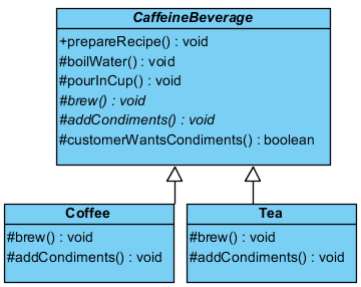
Maar de high-levelcomponent bepalen wanneer en hoe.

Low-levelcomponenten kunnen in de berekening participeren

Een low-levelcomponent roept nooit rechtstreeks een high-levelcomponent aan.

Het Hollywoodprincipe en de Template methode

Caffeinebeverage is onze high level component. Het bestuurt het algoritme voor het recept, en roept de subklassen alleen aan wanneer ze nodig zijn voor de implementatie van een methode.

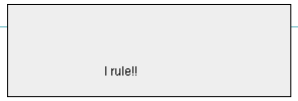


De 'clients' van de dranken zijn afhankelijk van de abstrahering in Caffeinebeverage en niet van de concrete klassen 'Tea' en 'Coffee'. Hierdoor neemt de afhankelijkheden in het hele systeem af.

De subklassen worden gewoon gebruikt om de implementatiedetails te leveren.
Tea en Coffee roepen de abstracte klasse nooit rechtstreeks aan zonder eerst te zijn 'aangeroepen'.

Template methode in het echt

- Swingen met frames
  - We breiden JFrame uit, die een update()-methode bevat die het algoritme bestuurt voor het updaten van het scherm. We kunnen in dit algoritme inhaken door een override van de hookmethode paint().




```
public class MyFrame extends JFrame {
        public MyFrame(String title) {
          super(title);
          this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
          this.setSize(300,300);
          this.setVisible(true);
        }
        public void paint(Graphics graphics) {
          super.paint(graphics);
          String msg = "I rule!!";
          graphics.drawString(msg, 100, 100);
        }…
```

Het update algoritme van JFrame roept paint() aan. Standaard doet paint() niets. Het is een hook. Door de override zorgen we ervoor dat een boodschap in venster getekend wordt

Template methode in het echt

- Applets
  - Iedere applet moet een subklasse van Applet zijn, Applet levert verscheidene hooks: init, start, paint, stop en destroy.

```
public class MyApplet extends JApplet {
  private String message;
   public void init() {
     message = "Hello World, I'm alive!";  repaint();
   }
   public void start() {
     message = "Now I'm starting up...";  repaint();
  }
   public void stop() {
     message = "Oh, now I'm being stopped...";   repaint();
  }
   public void destroy() {
     message = "Goodbye, cruel world";   repaint();
  }
   public void paint(Graphics g) {
     g.drawString(message, 5, 15);
   }
}
```

repaint() is een concrete methode in de klasse JApplet die laat weten dat de applet opnieuw getekend moet worden.